**CS673 Software Engineering** 

**Fitfusion**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Haoran Zheng | Team Leader | *Haoran Zheng* | 10/12/2024 |
| Jiankun Dong | Design and Implementation Leader | *Jiankun Dong* | 10/12/2024 |
| Chengqin Li | QA Leader | *Chengqin Li* | 10/12/2024 |
| Yuhan Pan | Requirement Leader | *Yuhan Pan* | 10/12/2024 |
| Yunrui Huang | Configuration Leader | *Yunrui Huang* | 10/12/2024 |
| Hangqi Wu | Security Leader | *Hangqi Wu* | 10/12/2024 |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **V0.01** | **Jiankun Dong** | **10/13/2024** | **Updated Introduction and Software Architecture** |
|  |  |  |  |

[Introduction](#_heading=h.gjdgxs)

[Software Architecture](#_heading=h.30j0zll)

[Class Diagram](#_heading=h.1fob9te)

[UI Design (if applicable)](#_heading=h.3znysh7)

[Database Design (if applicable)](#_heading=h.2et92p0)

[Security Design](#_heading=h.tyjcwt)

[Business Logic and/or Key Algorithms](#_heading=h.3dy6vkm)

[Design Patterns](#_heading=h.1t3h5sf)

[Any Additional Topics you would like to include.](#_heading=h.2s8eyo1)

[References](#_heading=h.17dp8vu)

[Glossary](#_heading=h.3rdcrjn)

# Introduction

This Software Design Document (SDD) outlines the architectural and design decisions for the FitFusion project, aimed at providing a comprehensive platform for fitness tracking and management. In response to the growing demand for effective workout solutions among fitness enthusiasts and beginners alike. This document serves as a guide to ensure a structured development of the software. The architecture is designed to be modular and scalable, ensuring a seamless user experience while allowing for future enhancements.

# Software Architecture

**The FitFusion system comprises the following key components:**

* Frontend:
  + Description: The user interface component where users interact with the application
  + Technology stack: Vue.js
  + Key Features:
    - User Registration and Login
    - Personalized Fitness Plan Management
    - Exercise Tracking Interface
    - Push Notifications for Reminders
* Backend:
  + Description: The server component that processes requests from the frontend and manages data storage and business logic.
  + Technology Stack: Node.js, Express.js
  + Key Features:
    - API endpoints for user authentication and data management
    - Business logic for exercise tracking and workout suggestions
    - Integration with third-party APIs for AI-driven recommendations
* Database:
  + Description: The persistent storage for user data, fitness tracking information, system configurations and supported exercise.
  + Technology Stack: MongoDB
  + Key Features:
    - User data management (registration, authentication)
    - Fitness data storage (exercise logs, plans)
    - Performance tracking data

**Component Relationships:**

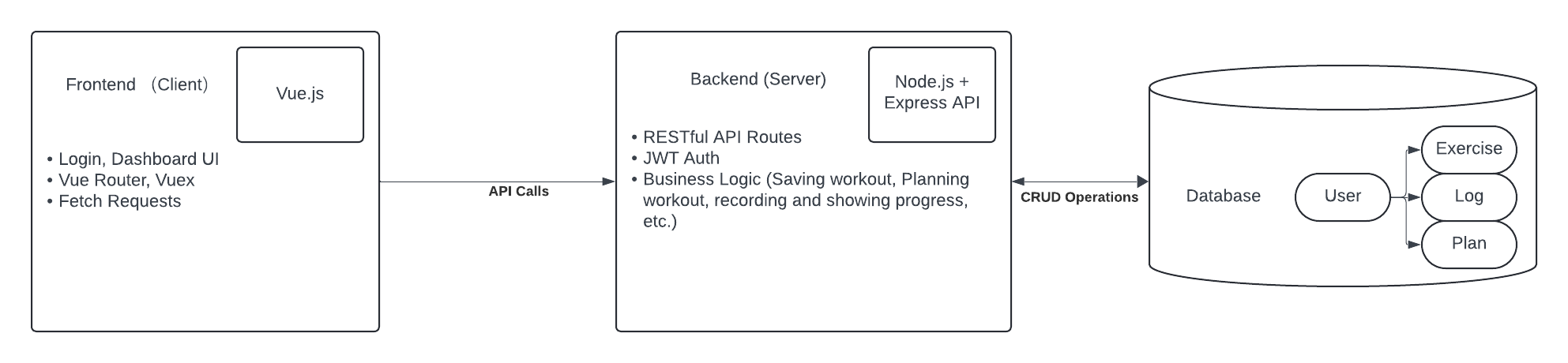
* Frontend to Backend: The frontend communicates with the backend via RESTful API endpoints, This communication allows the frontend to perform actions such as user registration and exercise logging.
* Backend to Database: The backend uses an Object Data Modeling (ODM) library, the Mongoose, to perform CRUD operations with the MongoDB database. This ensures efficient data management for users and their fitness data.

**Interface and dependencies:**

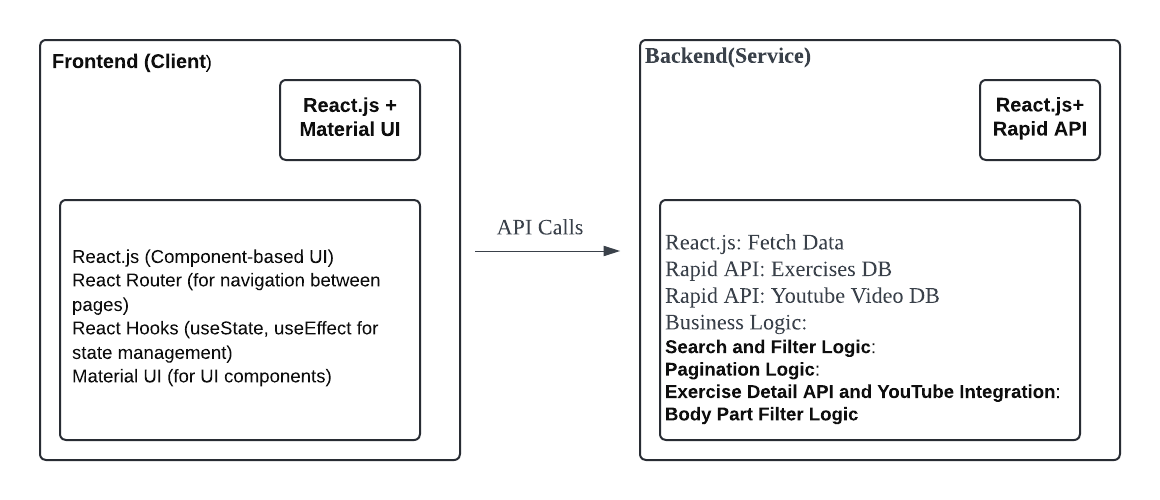
* Frontend Interfaces:
  + User Registration API:
    - Endpoint: POST /api/register
    - Description: Registers a new user
  + Exercise Tracking API:
    - Endpoint: GET /api/exercises
    - Description: Retrieves the list of logged exercises.
* Backend Interfaces:
  + User Management:
    - User.create(): Adds a new user to the database.
    - User.find(): Retrieves user information.
  + Exercise Management:
    - Exercise.create(): Logs new exercise data.
    - Exercise.find(): Fetches exercise records for a specific user.

**Frameworks and Technologies:**

* Frontend Framework: Vue.js
* Backend Framework: Node.js, Express.js
* Database Management: Mysql

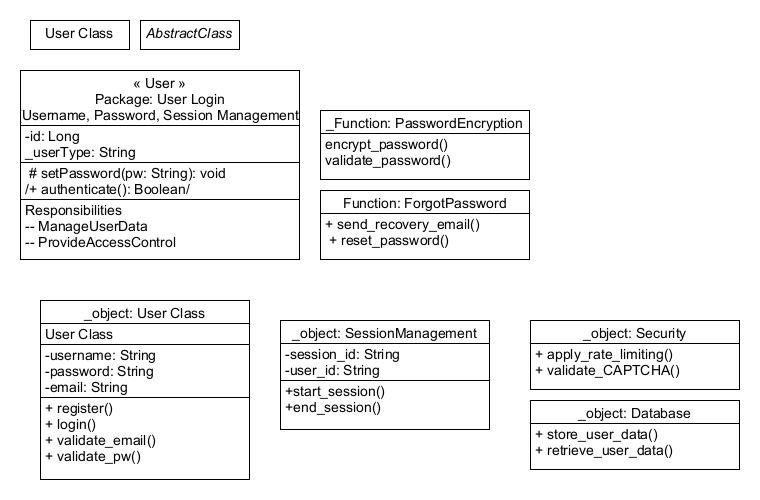
**Architecture Diagram**

# 

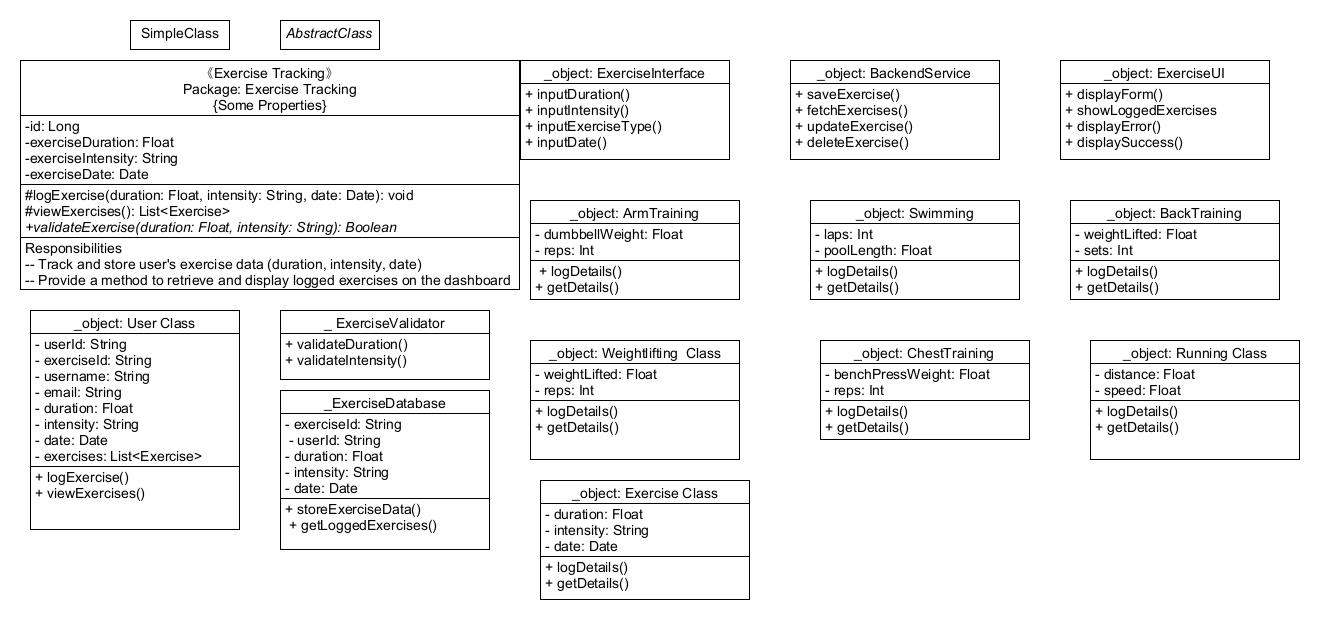


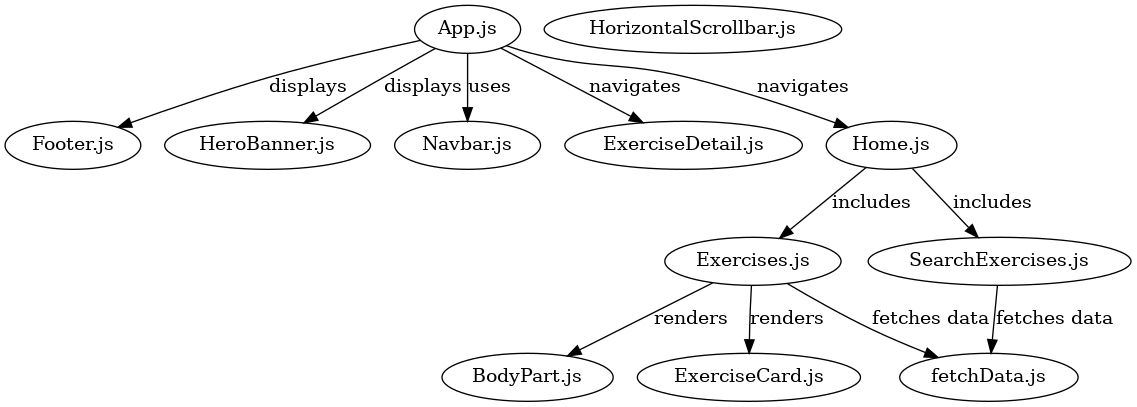
# Class Diagram

In this section, you will provide a detailed description of each component (or package) and use one or multiple class diagrams to show the main classes and their relationships in each component.

User Registration and Login

Exercise Tracking



Video and Illustration

# UI Design (if applicable)

In this section, you can describe your UI design

<https://mui.com/material-ui/getting-started/>

In the gym\_exercises UI design, I utilized Material UI components such as Stack, Box, Typography, and Button to create a clean and responsive layout:

• **Stack**: Used for arranging elements either vertically or horizontally, with automatic spacing between them. This helps streamline the layout without needing manual margin adjustments. For instance, a row of buttons or a column of text and input fields can be easily aligned.

• **Box**: A versatile layout component that serves as a flexible container for styling. It allows for control over positioning, padding, margins, and responsive behavior. It’s perfect for wrapping and positioning individual elements or groups of elements.

• **Typography**: This component is used to handle text styles and formatting. It ensures consistent font sizes, weights, and alignments across the application, making the text easy to read and visually appealing.

• **Button**: A highly customizable interactive component that facilitates user actions. Buttons can be styled in various ways (contained, outlined, text) to suit different use cases, such as primary actions or secondary links.

# Database Design (if applicable)

In this section, you shall describe any database schema if used in your software system.

User account:

User id int unique Auto increment

User Email char[] Primary Key //Frontend

User name char[] unique //Frontend

User password char[] (after Hash) //Frontend

User group id int (mult)

User role int

User Profile:

User id Primary Key

User full name

Exercise Recording:

User id //Primary Key from user account

Record id int Auto increment unique

exercise name char[] //以后拆分到额外table，部位和动作分类

Description char[]

Number of set int

Status //Todo/Doing/Done char[10]

Priority //0-10 int

Start time timestamp

End time timestamp

Total time //min by int

Future Plan:

Plan Id int unique Auto increment Primary Key

User Id

Exercise name varchar

Description varchar

Sets int

Date varchar

Priority //0-10 int

# Security Design

In this section, you shall describe any security design in your software system. Our system incorporates essential security measures to protect user data, particularly focusing on password management and authentication:

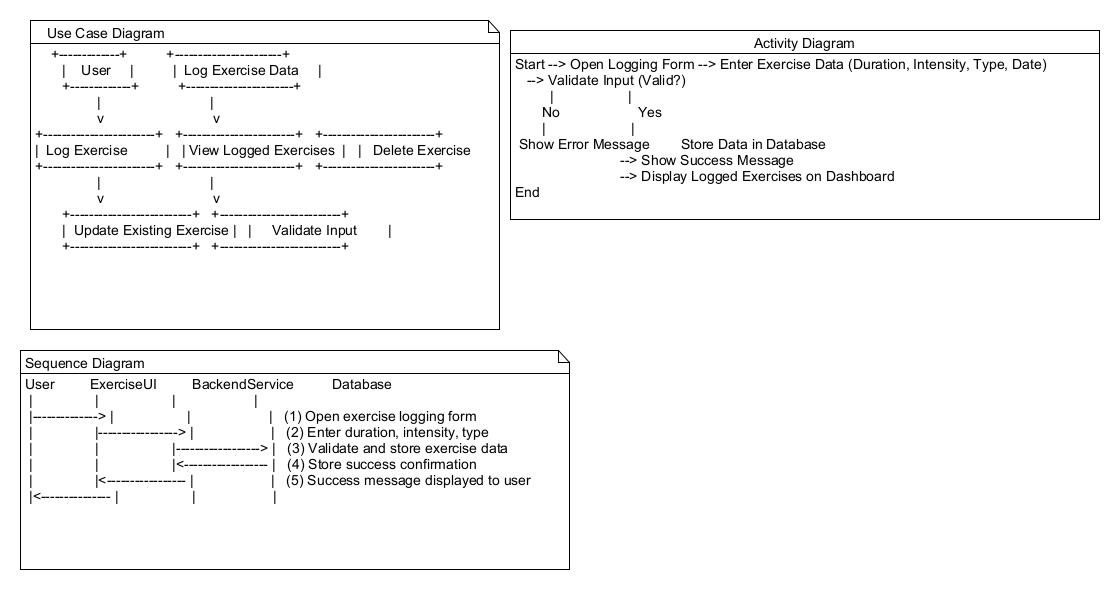
**1. Password Verification and Encryption**

* **Password Hashing**: User passwords are hashed before storage, ensuring they are never stored as plain text. Bcrypt adds a random **salt** to each password, making it resistant to brute-force attacks and ensuring unique hashes even for identical passwords.
* **Password Verification**: During login, the provided password is hashed again and compared with the stored hash to authenticate users securely.

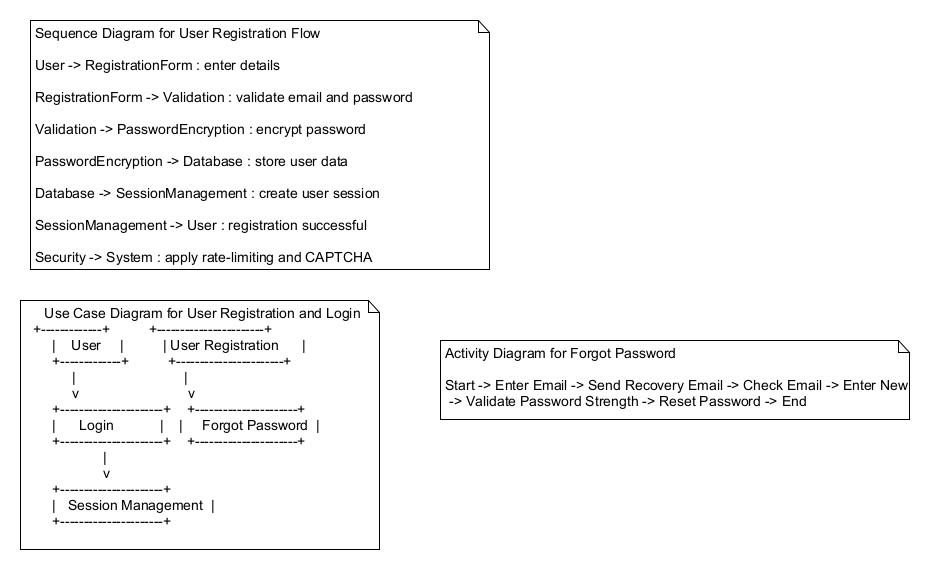
# Business Logic and/or Key Algorithms

In this section, you shall describe any key algorithms used in your software system, either in terms of pseudocode or flowchart, or sequence diagrams.

Exercise Tracking

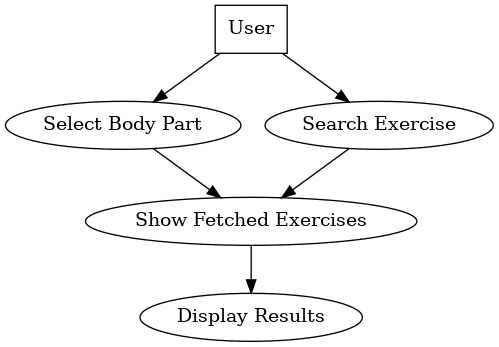


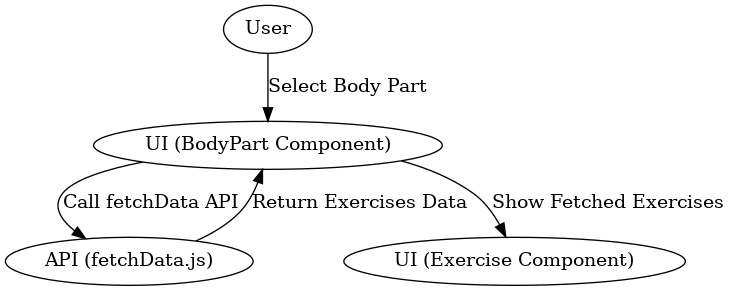
User Registration Flow



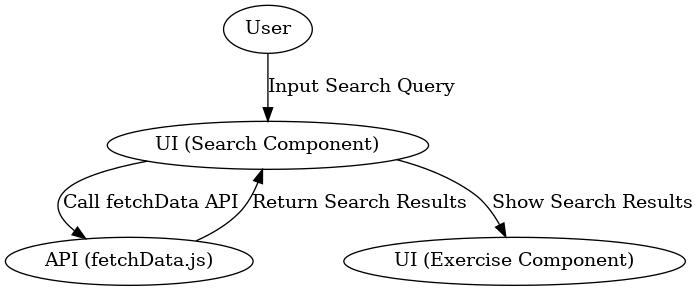
Video and Illustration

**Sequence Diagram (Select Body Part and Fetch Exercises)**:





**Sequence Diagram 2 (Search Exercises and Show Results)**:



# Design Patterns

In this section, you shall describe any design patterns used in your software system.

# Rest APIs

* Create New Account
  + End Point: ~/api/account
  + method: POST
  + This endpoint creates a new user account with the provided email, name, and password. If any required fields are missing or invalid, an error is returned.
  + Request body:

{ "email": "string (required)",

"name": "string (required)",

"password": "string (required)" }

* + Response:
    - Success (200 OK): { "token": "string (user ID or authentication token)" }
    - Error (400 Bad Request): { "err": "missing not null value" }
* **Login Account**
  + End Point: ~/api/account
  + method: GET
  + This endpoint authenticates a user based on their email or username and password. The user can log in with either their email or username
  + Request query:

{ "email": "string (optional)",

"name": "string (optional)",

"password": "string (required)" }

* + Response:
    - Success (200 OK): { "token": "string (user ID or authentication token)" }
    - Error (400 Bad Request): { "err": "missing not null value" }
* **Create New Exercise Record**
  + End Point: ~/api/record
  + method: POST
  + This endpoint allows users to create a new exercise record. The user must provide a valid token to authenticate.
  + Request body:

{ "token": "string (required)",

"exercise\_name": "string (required)",

"description": "string (optional)",

"number\_of\_set": "number (required)",

"status": "string (required)",

"priority": "string (required)",

"start\_time": "string (ISO 8601 format, required)",

"end\_time": "string (ISO 8601 format, required)",

"total\_time": "number (required)"}

* + Response:
    - Success (200 OK): { "result": "object (database response)" }
    - Error (400 Bad Request): { "err": "missing token" }
* **Delete Exercise Record**
  + End Point: ~/api/record
  + method: DELETE
  + This endpoint allows users to delete an exercise record. A valid token and record ID must be provided
  + Request query:

{ “token": "string (required)",

"record\_id": "string (required)" }

* + Response:
    - Success (200 OK): { "result": "object (database response)" }
    - Error (400 Bad Request): { "err": "missing token" }
* **Modify Exercise Record**
  + End Point: ~/api/record
  + method: PUT
  + This endpoint allows users to modify an existing exercise record. A valid token and record ID must be provided.
  + Request body:

{ "token": "string (required)",

"record\_id": "string (required)",

"exercise\_name": "string (required)",

"description": "string (optional)",

"number\_of\_set": "number (required)",

"status": "string (required)",

"priority": "string (required)",

"start\_time": "string (ISO 8601 format, required)",

"end\_time": "string (ISO 8601 format, required)",

"total\_time": "number (required)" }

* + Response:
    - Success (200 OK): { "result": "object (database response)"}
    - Error (400 Bad Request): { "err": "missing token" }
* Create New Account
  + End Point: ~/api/record
  + method: GET
  + This endpoint allows users to retrieve all exercise records associated with their user ID. A valid token must be provided.
  + Request body:

{ "token": "string (required)"}

* + Response:
    - Success (200 OK): { "result": "array of objects (exercise records)" }
    - Error (400 Bad Request): { "err": "missing token" }

Rapid APIS:

https://rapidapi.com/justin-WFnsXH\_t6/api/exercisedb

**1. Exercise Database API**

* The **exerciseOptions** object configures the request to fetch exercise data. It uses the X-RapidAPI-Host and X-RapidAPI-Key headers for authentication and authorization with the ExerciseDB API. The data fetched from this API includes a list of exercises, body parts, equipment, and targeted muscle groups.

export const exerciseOptions = {

method: 'GET',

headers: {

'X-RapidAPI-Host': 'exercisedb.p.rapidapi.com',

'X-RapidAPI-Key': 'YOUR\_RAPIDAPI\_KEY',

},

};

**2. Data Fetching Function**

The fetchData function is a reusable utility that takes a URL and options object to make API requests using the Fetch API. It abstracts the logic for making GET requests, making it easy to reuse across different API calls.

export const fetchData = async (url, options) => {

const res = await fetch(url, options);

const data = await res.json();

return data;

};

# Any Additional Topics you would like to include.

# References

# Glossary